Diagramme de features, adaptation par transformation dans le contexte des bâtiments intelligents

Thibaut Possompès[1,2], François Briant[1], Christophe Dony[2], Marianne Huchard[2], and Chouki Tibermacine[2]

[1] IBM France – PSSC Montpellier
thibaut.possompes,fbriant@fr.ibm.com
[2] LIRMM, CNRS and Université de Montpellier 2
possompes,dony,huchard,chouki.tibermacine@lirmm.fr

Résumé Les lignes de produits appliquées aux *bâtiments intelligents* requièrent de larges modèles de features dans lesquels la variabilité exprimée reflète l'instrumentation et la composition du bâtiment. De plus, certains points de variabilités doivent être dupliqués sur chaque occurrence de certains éléments du bâtiments (par exemple, des scénarios d'optimisation dédiés à chaque pièce du bâtiment). Nous proposons une approche dirigée par les modèles permettant, à partir d'un diagramme de features reflétant le contrôle énergétique dans toute sa globalité, de générer un diagramme de features spécifique à un bâtiment donné. C'est à dire, un diagramme n'incluant que les options appropriées relatives au bâtiment à équiper. Ce papier présente comment un modèle d'infrastructure (décrivant un bâtiment), et un modèle de features sont transformés pour générer un modèle de feature spécifique à un bâtiment.

mots-clefs : Diagrammes de features, profils UML, lignes de produits logicielles, ingnériérie dirigée par les modèles, bâtiments intelligents

Abstract Software product lines in *smart buildings* involve large feature models in which the variability reflects the instrumentation and composition of the building. Moreover, some variation points must be duplicated on each occurrence of certain elements of the building (*e.g.*, optimization scenarios dedicated to each room of the building). We propose a model-driven approach to generate and configure feature models specific to a given building. In other ways, a diagram showing only appropriate features for a given building. This paper presents how an infrastructure model (describing a building), and a feature model are transformed to generate a infrastructure-specific feature diagrams.

keywords : Feature diagrams, UML profiles, software product lines, model-driven engineering, smart buildings

1 Introduction

L'approche par ingénierie dirigée par les modèles présentée dans cette publication vise la mise en oeuvre de lignes de produits logicielles dans le contexte du projet RIDER [3].

La plate-forme RIDER vise à permettre la gestion de la consommation énergétique à différents niveaux d'échelles, dans un complexe de bâtiments utilisant des standards, et des modes d'instrumentation différents. A l'échelle locale, RIDER vise à améliorer la consommation énergétique d'un bâtiment en centralisant des sources d'informations diverses, telles que les températures intérieures et extérieures, les consommations électriques, l'architecture du bâtiment, l'occupation des locaux, afin d'émettre des recommandations permettant un usage plus efficace des ressources énergétiques disponibles. Notre objectif final consiste à permettre la personnalisation et la génération de la plate-forme RIDER conformément aux besoins du client et les caractéristiques de ses bâtiments.

Cet article décrit une méthode permettant de personnaliser la composition d'un produit RIDER en fonction du modèle du bâtiment à gérer. Par exemple, afin d'utiliser des scénarios d'optimisation différents dans chaque pièce. L'utilisation de modèles de features tels que décrits dans la littérature [1,2,3,4,5,6,7] ne permet pas d'obtenir ce niveau de personnalisation. Notre problématique nécessite d'avoir des diagrammes de features spécifiques à chaque modèle de bâtiment. C'est pourquoi nous avons développé une méthode permettant d'adapter un modèle de features générique par rapport à un modèle de bâtiment fournit par le commanditaire d'un produit RIDER.

La section 2 présente les modèles utilisés pour représenter des bâtiments et la ligne de produit logicielle. La section 3 décrit comment les modèles sont combinés pour générer un diagramme de feature spécifique à un bâtiment donné. La section 4 conclue cet article.

2 Modélisation de bâtiments et variabilité

La création d'un logiciel d'optimisation énergétique de bâtiments requiert la capacité de modéliser des bâtiments, et de s'adapter aux spécificités de chacun ainsi que leurs évolutions. Nous identifions comme étant *featurables*, les éléments du modèle du bâtiment nécessitant une adaptation du produit, au moyen d'un modèle de features spécifique. Pour répondre à ces besoins, nous avons conçu 3 méta-modèles différents :

- Méta-modèle d'infrastructure : Fournit les éléments permettant la description de l'infrastructure d'un site, de bâtiments, pièces, l'instrumentation (capteurs, actionneurs), *etc.*

3. Le projet RIDER ("Research for IT as a Driver of EneRgy efficiency") est mené par un consortium de différentes sociétés et laboratoires de recherche, dont IBM et le LIRMM, ayant pour objectif commun d'améliorer l'efficacité énergétique des bâtiments.

- Profil UML d'éléments *featurables* : Permet de marquer les classes du méta-modèle d'infrastructure pour lesquelles le modèle de features devra être adapté
- Profil UML permettant la création de modèles de features : Comme décrit dans [8], ce profil a été mis en oeuvre sous la forme d'un plugin Rational Software Architect (RSA) afin de réaliser des modèles de features.

Le méta-modèle d'infrastructure sert à décrire l'ensemble des données manipulées par une plate-forme RIDER. Une instance de ce méta-modèle, modélise un bâtiment ou groupe de bâtiments. Les classes du méta-modèle d'infrastructure peuvent être stéréotypées à l'aide du profil d'éléments featurables. Chaque instance d'élément featurable peut être personnalisé individuellement au moyen d'un diagramme de features dédié. Ainsi, le modèle de features d'un bâtiment sera généré en fonction du modèle du bâtiment, et à partir d'un modèle de features type pour RIDER. Le modèle de features d'un bâtiment est généré à l'aide d'associations, appelées *Model Relationships*, entre des features du modèle de features type pour RIDER, et des éléments featurables du méta-modèle d'infrastructure.

Cette approche nous permet de personnaliser le comportement d'un produit (une instance de la plate-forme RIDER générée pour un client) d'une façon spécifique à chaque bâtiment.

2.1 Mise en oeuvre du méta-modèle de bâtiments

Le méta-modèle d'infrastructure a pour objectif centraliser la modélisation de bâtiments au sein du projet RIDER. Il comporte environ 150 classes permettant notamment de modéliser des plan de bâtiments, les capteurs et actionneurs, et différents indicateurs permettant l'utilisation d'algorithmes d'optimisation. Il permet de faire abstraction de la complexité du bâtiment en permettant la modélisation de concepts de haut niveau, tels que des services de climatisation, ou des systèmes d'instrumentation.

Le méta-modèle peut être instancié de différentes manières : manuellement à l'aide d'un éditeur approprié, automatiquement à partir d'une source de données telle qu'un logiciel de supervision de bâtiments ou d'une maquette numérique [9,10], ou algorithmiquement.

2.2 Éléments *featurables*

Certaines classes du méta-modèle d'infrastructure peuvent induire une adaptation du modèle de features de la ligne de produits RIDER. Par exemple, certains scénarios d'économie d'énergies ne pourront être associés qu'à certaines pièces ou bâtiments, un data center peut faire l'objet d'analyses spécifiques telles que la répartition de la charge sur les serveurs, ou l'extraction de son énergie fatale[4]. Les éléments *featurables* servent donc à marquer les classes du méta-modèle d'infrastructure dont l'instance peut-être un point de variabilité.

4. Chaleur produite par le fonctionnement du data centre, et pouvant être réutilisée.

Méta-modèle La figure 1 présente le méta-modèle des éléments featurable. Il s'agit d'une structure de données en arborescence permettant de représenter une ou plusieurs hiérarchies d'éléments. Chaque *Featurable Element* comporte l'identifiant et le nom d'une classe du méta-modèle d'infrastructure. L'attribut *instances* contient la liste des instances présentes dans le modèle d'un bâtiment. La classe *Element Link* représente un lien hiérarchique entre deux *Featurable Elements*. Les attributs *parentFieldName* et *childFieldName* sont utilisés pour identifier la source et la destination de l'association entre deux *Featurable Elements*.

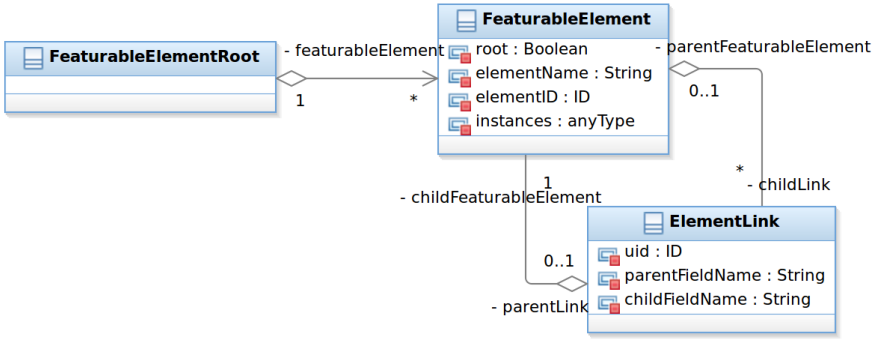

Figure 1. Méta-modèle des éléments *featurables*

Profil UML Nous avons dérivés le méta-modèle décrit précédemment en un profil UML, présenté dans la Figure 2. Nous retrouvons les stéréotypes *FeaturableElement* et *ElementLink*, et nous avons ajouté *FeaturableElementLinkDirection* afin de décrire la direction de l'association.

La figure 3 présente un exemple d'utilisation d'éléments featurables représentant la hiérarchie d'un site, contenant des bâtiments, lesquels contiennent des étages constitués d'espaces qui eux-mêmes peuvent être décomposés en sous-espaces. Le comportement de RIDER pourra être paramétré pour chaque occurrence de l'un de ces éléments chez un client.

2.3 Ligne de produits logicielle

Nous utilisons le plug-in pour Rational Software Architect présenté en [8] pour créer nos modèles de features à l'aide d'un profil UML. Cette approche permet de facilement associer des features à des éléments featurables du méta-modèle d'infrastructure en utilisant un outil et un langage de modélisation commun. Ainsi, une feature est associée, graphiquement, à un élément featurable par

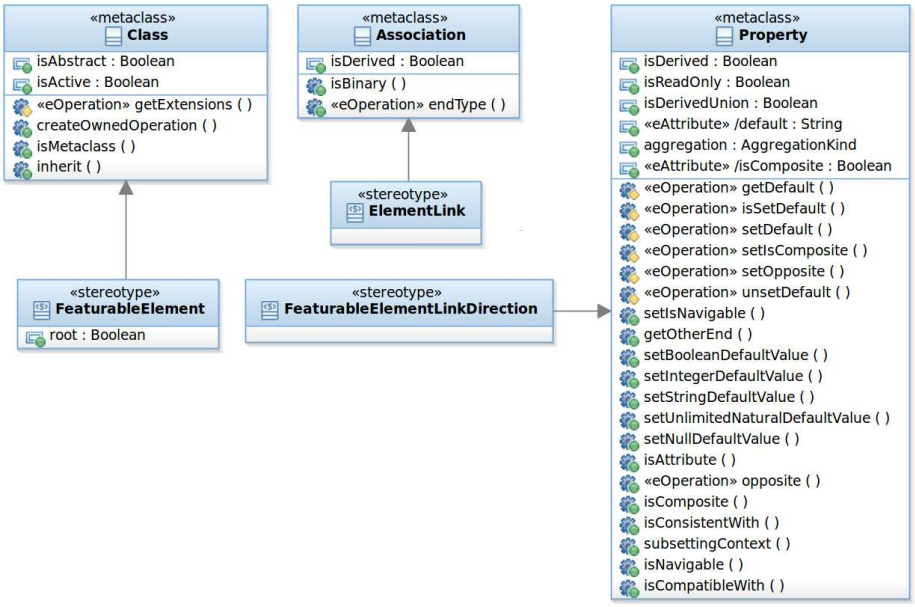

FIGURE 2. Profil des éléments *featurables*

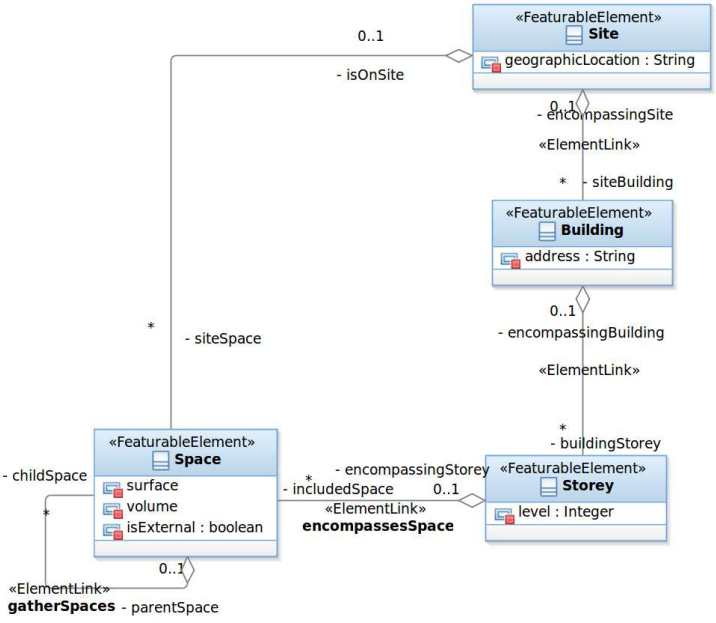

FIGURE 3. Exemple d'éléments *featurables*

une association UML. Le diagramme de la Figure 4 présente un extrait du modèle de feature de RIDER dans lequel la feature *Standard building scenarios* est associée à la classe featurable *Building*, issue du méta-modèle d'infrastructure. Ce diagramme a été réalisé avec notre plug-in RSA.

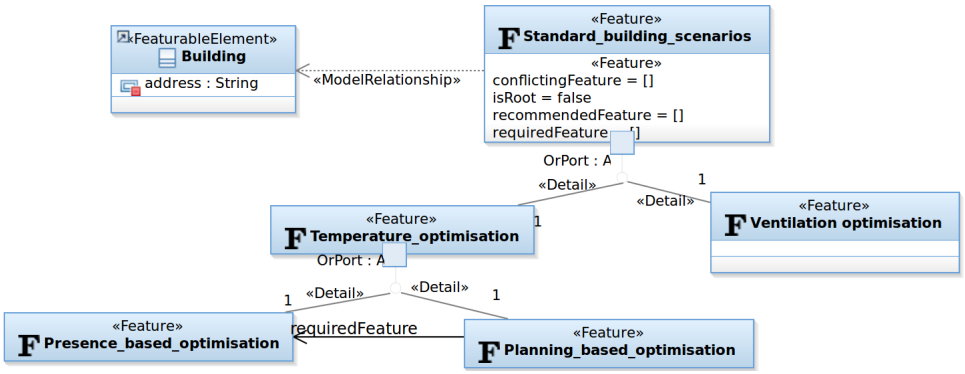

Figure 4. Association entre feature et élément featurable

3 Génération de diagrammes de features spécifiques à un bâtiment

La Figure 5 présente une vision globale des modèles utilisés dans notre approche. Le diagramme est divisé en deux parties distinctes correspondant au framework de lignes de produits décrit en [1]. La partie *I* étant le *Domain Engineering*, et la partie *II* l'*Application Engineering*. Ainsi, les modèles réalisés en *I* sont réutilisés pour chaque nouvelle configuration d'une plate-forme RIDER en fournissant le langage de description de bâtiments, les éléments featurable et un modèle de features générique. Les modèles utilisés en *II* sont spécifiques à chaque client.

Les étapes requises pour mettre en oeuvre notre approche sont les suivantes :

1. Pré-requis – Méta-modèle d'infrastructure et modèle de features
   (a) Le méta-modèle d'infrastructure est requis. Son rôle consiste à décrire les éléments qui pourront être soumis à des points de variabilités. Il s'agit du modèle de données de la plate-forme RIDER.
   (b) Identifier les éléments *featurables* dans le méta-modèle d'infrastructure à l'aide du profil UML des éléments *featurables*.
   (c) Un modèle de features réalisé à l'aide de notre profil UML [8].
      i. Si nécessaire des features peuvent être liés à une ou plusieurs classes du méta-modèle d'infrastructure.

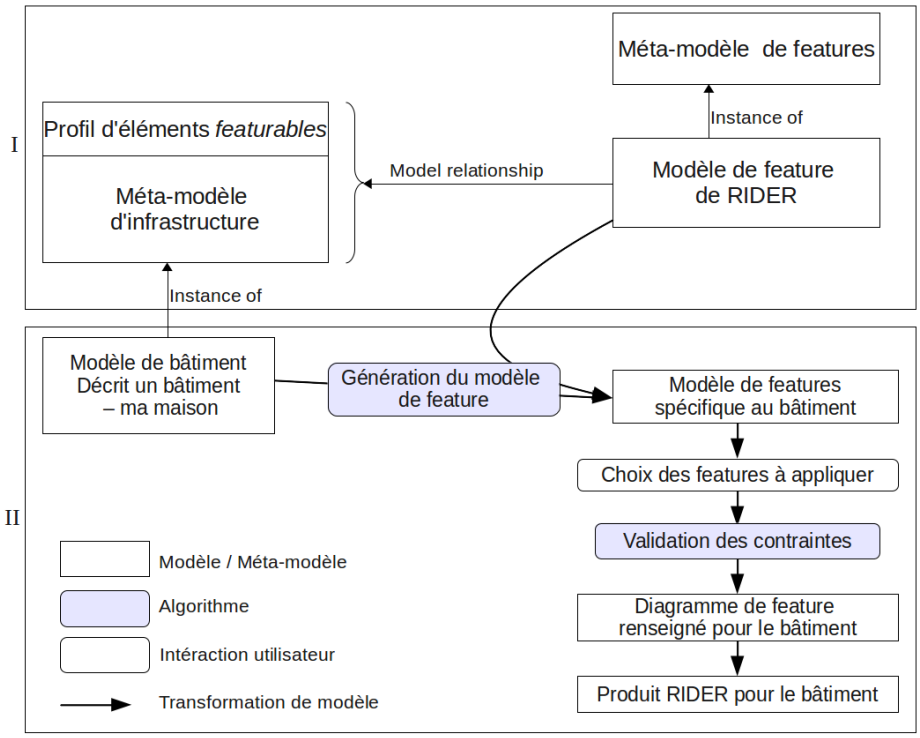

FIGURE 5. Modèle de feature généré par une approche dirigée par des modèles

ii. Chaque feature peut peut être associée à des règles métier permettant de déterminer si elle peut être sélectionnée dans le contexte d'un modèle d'infrastructure.

2. Adaptation au client.

   (a) Le client fournit la description d'un bâtiment ou site sur lequel RIDER doit être déployé – *i.e.*, une instance du méta-modèle d'infrastructure.

   (b) Génération du modèle de features correspondant au bâtiment décrit. Chaque feature est contrôlée pour déterminer à quels éléments du modèle du bâtiment elle peut être associée (en fonction de règles).

   (c) Choix des features par les personnes concernées. Des filtres sont utilisés pour afficher uniquement les features pertinantes pour une personne en fonction de son rôle.
   - Les features ne pouvant être sélectionnées sont présentées à l'utilisateur ainsi que les contraintes non satisfaites.
   - Si les features applicables au modèle métier ne conviennent pas, il est possible d'ajouter les concepts manquant dans le modèle d'infrastruc-

ture, en faisant les modifications qui s'imposent alors dans le bâtiments.
  - Le modèle du bâtiment doit représenter la réalité, mais dans le cas de l'instrumentation d'un nouveau bâtiment, l'installation des capteurs et actionneurs peut être faite à posteriori, conformément aux besoins de RIDER. Ainsi, si l'utilisateur ajoute des instruments nécessaires à certaines features, il est alors nécessaire de produire un rapport décrivant les besoins d'instrumentation de RIDER. A l'exception des instruments virtuels tous doivent être accessibles par l'intermédiaire d'un logiciel de supervision de bâtiment.
  - Par la suite l'utilisateur pourra avoir à ajouter ou retirer des features du produit déployé pour suivre les évolutions du bâtiment.

(d) Le produit est généré conformément au modèle d'infrastructure et aux features qui ont été renseignées.

i. L'asset associé à chaque feature contient un script permettant de déployer les éléments nécessaires à son fonctionnement.

ii. L'asset contient tous les éléments nécessaires à son fonctionnement, mais peuvent nécessiter ou être en conflit avec d'autres assets.

La Figure 6 présente une capture d'écran de l'outil que nous avons développé. La vue de gauche, nommée *Featurable elements*, présente quels sont les éléments du bâtiment pouvant être individuellement personnalisés. Compte tenu du fait que nous avions identifié dans le méta-modèle d'infrastructure, dans l'exemple de la Figure 3, les classes *Site, Building, Storey* et *Space* comme étant *Featurables*, nous retrouvons leurs instances dans cette vue.

La vue *Feature tree view* filtre les features affichables, *i.e.*, correspondant à l'élément featurable sélectionné dans la vue *Featurable elements*, parmi celles du diagramme de features ayant été généré.

3.1 Génération de la vue *Featurable elements*

Les algorithmes 1, 2 et 3 présentent comment la vue *Featurable elements* de la Figure 6 est générée. L'algorithme 1 initialise le modèle d'éléments featurables spécifique au modèle d'infrastructure fournit. Le modèle d'éléments featurables est extrait à partir des stéréotypes placés sur le méta-modèle d'infrastructure. L'algorithme associe une liste d'instances d'une méta-classe featurable, trouvées dans le modèle d'infrastructure, à l'élément featurable correspondant.

L'algorithme 2 crée les éléments qui seront disponibles dans la vue *Featurable elements*. Chaque élément featurable marqué *root*, sera la racine d'un arbre présenté dans cette vue. Dans l'exemple de la Figure 6, seule la méta-classe *Site* est *root*, et le modèle d'infrastructure fournit n'en comporte qu'une instance.

L'algorithme 3 insère un nouveau noeud dans l'arbre. Un noeud est constitué d'une référence vers un élément featurable, et une référence vers un élément du modèle d'infrastructure (*e.g.*, une pièce, un bâtiment donné). Chaque arbre de la

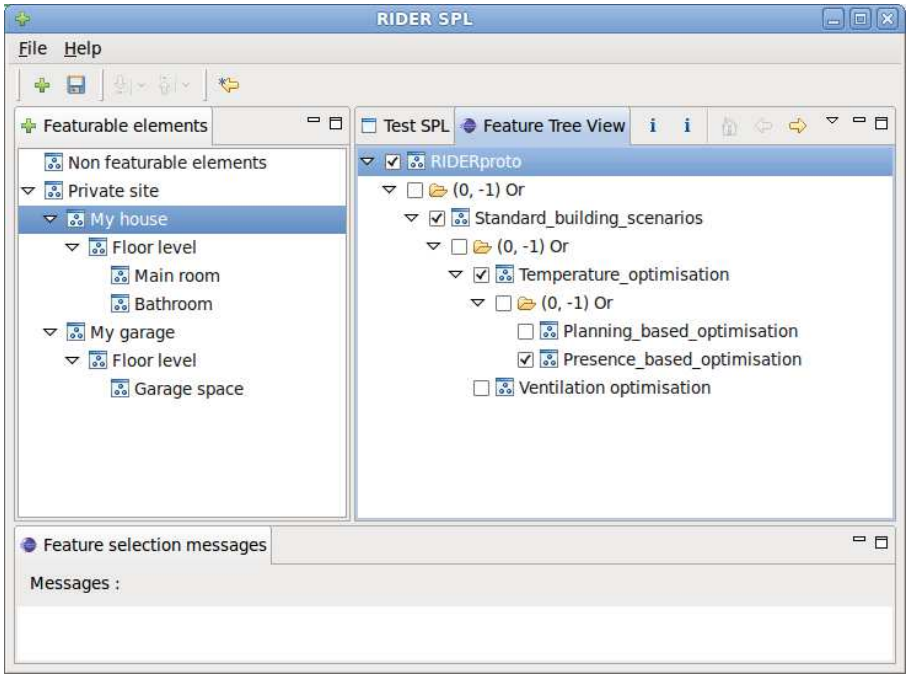

Figure 6. Prototype du configurateur

vue est conforme à la hiérarchie, décrite par les stéréotypes du profil *Featurable elements*, des classes du modèle d'infrastructure. Il s'agit d'un algorithme récursif appelé par l'algorithme 2

3.2 Génération de la vue *Feature tree view*

L'algorithme 4 présente le principe permettant de générer la vue *Feature tree view*. Nous n'avons pas décrit le détail de l'algorithme ici en raison de contraintes d'espace, il s'agit donc d'une version simplifiée. L'algorithme, récursif, doit être initialisé avec la feature racine du modèle de features de RIDER. Le modèle de features spécifique au modèle d'infrastructure résultant de cet algorithme

Data : modèle d'éléments featurable, modèle d'infrastructure
Result : Modèle d'éléments featurable initialisé, dans lequel chaque élément est associé aux classe correspondantes du modèle d'infrastructure
foreach *featurable element* **do**
  Cherche une instance de la méta-classe correspondante dans le modèle d'infrastructure et l'associe au *featurable element* courant;
end

Algorithme 1: Initialisation du modèle d'éléments featurables

```
Data : modèle d'éléments featurable initialisé, modèle d'infrastructure
Result : Arbres permettant de lister les éléments featurable du modèle
         d'infrastructure fournit par l'utilisateur
foreach featurable element do
    read current;
    if current is root then
        Appelle la fonction d'ajout d'un nouveau noeud dans l'arbre, algorithme
        3;
    end
end
```

Algorithme 2: Construction de l'arbre de la vue contenant les instances du méta-modèle d'infrastructure personnalisables

```
Data : un élément featurable et le noeud parent sous lequel il devra s'insérer
Result : Un noeud de l'arbre permettant de lister les éléments featurable du
         modèle d'infrastructure fournit par l'utilisateur
if l'élément featurable est associé à au moins une classe du modèle
d'infrastructure then
    foreach instance de l'élément featurable do
        if l'instance appartient à l'une des instances enfant de la classe du
        modèle d'infrastructure associée à l'élément featurable parent then
            Ajout d'un noeud dans l'arbre associant l'élément featurable avec
            l'instance;
        end
        foreach Element featurable enfant de l'élément featurable courant do
            Appeler la fonction d'ajout d'un nouveau noeud dans l'arbre;
        end
    end
end
```

Algorithme 3: Ajout d'un noeud dans l'arbre des éléments featurable d'un modèle d'infrastructure

est ensuite soumis à un filtre afin de n'afficher dans la vue, que les features relatives à l'élément featurable du modèle d'infrastructure ayant été sélectionné dans la vue *Featurable elements*. La vue présentée dans l'exemple de la Figure 6 montre également les features parentes des features liées à l'élément featurable sélectionné afin de simplifier à l'utilisateur la tâche de sélection des features. Une feature ne pouvant être choisie que si ses features parent le sont également, nous les affichons donc afin permettre à l'utilisateur de les sélectionner sans changer la vue.

```
Data : une feature issue du modèle de features de RIDER à associer au modèle
       de features spécifique, l'élément featurable du modèle d'infrastructure
       auquel est associée la feature parent spécifique au modèle
       d'infrastructure
Result : le modèle de features spécifique modèle d'infrastructure enrichi de la
         feature à ajouter
foreach classe du méta-modèle d'infrastructure auquel est associé la feature do
    if la feature n'est pas associée à un élément de l'arbre des instances
    d'éléments featurables then
        foreach élément de l'arbre des instance d'éléments featurables do
            if la feature courante peut être associée à cet élément then
                créer une feature spécifique à cet élément et l'insérer dans le
                modèle de feature spécifique;
            end
        end
    else
        créer une feature spécifique à la méta-classe de cet élément;
    end
end
if la feature n'est pas associée à une classe du méta-modèle d'infrastructure
then
    ajouter la nouvelle feature, sans l'associer à un élément de l'arbre des
    instance d'éléments featurables;
end
```

Algorithme 4: Ajout d'un noeud dans l'arbre des éléments featurable d'un modèle d'infrastructure

4 Conclusion et perspectives

Nous proposons une application de lignes de produits logicielles dont les diagrammes sont générés à l'aide d'une approche dirigée par les modèles. Ces travaux nous permettent de personnaliser les choix disponibles pour une plate-forme RIDER en fonction du site sur lequel elle sera déployée. Un prototype de cet outil a été développé et testé sur des extraits de la ligne de produits de RIDER. Nous avons implémenté également la vérification de contraintes sur le

diagramme de features spécifique au modèle d'infrastructure, vérifiées à mesure que les features sont sélectionnées. Toute erreur est affichée dans la vue *Feature selection messages*, visible dans la Figure 6. Nous projetons d'ajouter prochainement une validation de la structure du modèle d'infrastructure à l'aide de règles, afin de filtrer plus précisément les features proposées à l'utilisateur, et un solveur de contraintes tel que décrit par exemple par [11] afin d'éviter toute erreur de configuration.

Références

1. Pohl, K., Böckle, G., Linden, F.v.d. : Software Product Line Engineering : Foundations, Principles, and Techniques. Springer (2005)
2. Haugen, O., Moller-Pedersen, B., Oldevik, J., Olsen, G.K., Svendsen, A. : Adding standardized variability to domain specific languages. In : Proceedings of the 2008 12th International Software Product Line Conference, IEEE Computer Society (2008) 139–148
3. Morin, B., Perrouin, G., Lahire, P., Barais, O., Vanwormhoudt, G., Jézéquel, J. : Weaving variability into domain metamodels. In Schürr, A., Selic, B., eds. : Model Driven Engineering Languages and Systems. Volume 5795 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2009) 690–705 10.1007/978-3-642-04425-0_56.
4. Perrouin, G., Vanwormhoudt, G., Morin, B., Lahire, P., Barais, O., Jézéquel, J. : Weaving variability into domain metamodels. Software and Systems Modeling (2010) 1–23 10.1007/s10270-010-0186-4.
5. Acher, M., Collet, P., Fleurey, F., Lahire, P., Moisan, S., Rigault, J.P. : Modeling context and dynamic adaptations with feature models. In : 4th International Workshop Models@ run. time at Models. Volume 9. (2009)
6. Fernandes, P., Werner, C., Murta, L. : Feature modeling for context-aware software product lines. In : Proceedings of the 20th International Conference on Software Engineering & Knowledge Engineering (San Francisco, CA, USA, 2008). (2008)
7. Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J.M., Solberg, A., Dehlen, V., Blair, G. : An aspect-oriented and model-driven approach for managing dynamic variability. Model Driven Engineering Languages and Systems (2010) 782–796
8. Possompès, T., Dony, C., Huchard, M., Tibermacine, C. : Design of a UML profile for feature diagrams and its tooling implementation. In : Software Engineering and Knowledge Engineering (SEKE 2011), Miami, FL, USA (2011)
9. : The IFC specification — buildingSMART technical resources. http ://www.iai-tech.org/products/ifc_specification
10. Steel, J., Drogemuller, R., Toth, B. : Model interoperability in building information modelling. Software and Systems Modeling (SoSyM) 1–11 10.1007/s10270-010-0178-4.
11. White, J., Schmidt, D.C., Benavides, D., Trinidad, P., Ruiz-Cortés, A. : Automated diagnosis of Product-Line configuration errors in feature models. In : Proceedings of the 2008 12th International Software Product Line Conference, Washington, DC, USA, IEEE Computer Society (2008) 225–234